

RANDOM FORESTS FOR BIG DATA

Robin Genuer¹, Jean-Michel Poggi², Christine Tuleau-Malot³ & Nathalie Villa-Vialaneix⁴

¹ *INRIA, SISTM team & ISPED, INSERM U-897, Univ. Bordeaux, France*

robin.genuer@isped.u-bordeaux2.fr

² *LMO, Univ. Paris-Sud Orsay & Univ. Paris Descartes, France*

jean-michel.poggi@math.u-psud.fr

³ *Lab. Jean-Alexandre Dieudonné, Univ. Nice - Sophia Antipolis, France*

malot@unice.fr

⁴ *INRA, UR 0875 MIAT, 31326 Castanet Tolosan cedex, France*

nathalie.villa@toulouse.inra.fr

Abstract. Big Data is one of the major challenges of statistical science and has numerous consequences from algorithmic and theoretical viewpoints. Big Data always involve massive data but they also often include data streams and data heterogeneity. Recently some statistical methods have been adapted to process Big Data, like linear regression models, clustering methods and bootstrapping schemes. Based on decision trees combined with aggregation and bootstrap ideas, random forests were introduced by Breiman in 2001. They are a powerful nonparametric statistical method allowing to consider in a single and versatile framework regression problems, as well as two-class and multi-class classification problems. Focusing on classification problems, this paper reviews available proposals about random forests in parallel environments as well as about online random forests. Then, we formulate various remarks for random forests in the Big Data context. Finally, we experiment three variants involving subsampling, Big Data-bootstrap and MapReduce respectively, on two massive datasets (15 and 120 millions of observations), a simulated one as well as real world data.

Keywords. Big Data, MapReduce, Parallel Computing, Random Forests

1 Introduction

Big Data is one of the major challenges of statistical science and a lot of recent references start to think about the numerous consequences of this new context from the algorithmic viewpoint and for the theoretical implications of this new framework (see [1, 2, 3]). Big Data always involve massive data but they also often include data streams and data heterogeneity (see [4] for a general introduction), leading to the famous three V (Volume, Velocity and Variety) highlighted by the Gartner, Inc., the advisory company about information technology research¹. The problem posed by this large amount of data is twofold: first, as many statistical procedures have devoted few attention to computational runtimes, they can take too long to provide results in an acceptable time. Second, when the data are particularly massive, they can even be too big to hold in a single computer's memory. As pointed out in [3], in the near

¹<http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf>

future, statistics will have to deal with problems of scale and computational complexity issues to remain relevant. In particular, the collaboration between statisticians and computer scientists is needed to control runtimes that will maintain the statistical procedures usable on large-scale data while ensuring good statistical properties.

Recently some statistical methods have been adapted to process Big Data, including linear regression models, clustering methods and bootstrapping schemes. See [5] and [6] for recent reviews and useful references. The main proposed strategies are based on i) *subsampling* [7, 8, 9, 10, 11], ii) *divide and conquer approaches* [12, 13, 14], which consist in splitting the problem into several smaller problems and in gathering the different results in a second and final step, iii) *algorithm weakening* [15], which explicitly treats the trade-off between computational time and statistical accuracy using a hierarchy of methods with increasing complexity, iv) *online updates* [16, 17], which update the results with sequential steps, each having a low computational cost. However, only a few papers really address the question of the difference between the “small data” standard framework compared to the Big Data in terms of statistical accuracy. Noticeable exceptions are the article of Kleiner *et al.* [9] who prove that their “bag of little bootstrap” method is statistically equivalent to the standard bootstrap, the article of Chen and Xie [13] who demonstrate asymptotic equivalence to their “divide-and-conquer” based estimator with the estimator based on all data in the setting of regression and the article of Yan *et al.* [8] who show that the mis-clustering rate of their subsampling approach, compared to what would have been obtained with a direct approach on the whole dataset, converges to zero when the subsample size grows (in an unsupervised setting).

Based on decision trees and combined with aggregation and bootstrap ideas, random forests (abbreviated RF in the sequel), were introduced by Breiman [18]. They are a powerful nonparametric statistical method allowing to consider regression problems as well as two-class and multi-class classification problems, in a single and versatile framework. The consistency of RF has recently been proved by Scornet *et al.* [19], to cite the most recent result. On a practical point of view, RF are widely used (see [20, 21] for recent surveys) and exhibit extremely high performance with only a few parameters to tune. Since RF include intensive resampling and parallel construction of a lot of models (the trees of a given forest), it is natural to think about using parallel processing and to consider adapted bootstrapping schemes for massive data streams. The method has already been adapted and implemented to handle Big Data in various distributed environment (see, for instance, the libraries Mahout² or MLlib, the latter for the distributed framework “Spark”³, among others).

In this paper, we do not seek to make an exhaustive description of the various implementations of RF in scalable environments but we will review the problems posed by the Big Data framework, explain how those are usually handled and propose alternatives to help improve the relevance and accuracy of the approach in scalable environments. Focusing on classification problems, we then formulate various remarks for RF in the Big Data context. We finally experiment on two massive datasets (a simulated one and a real world one), three variants of RF for Big Data involving subsampling, Big Data-bootstrap and MapReduce respectively.

²<https://mahout.apache.org>

³<https://spark.apache.org/mllib>

The paper is organized as follows: after this introduction, we briefly recall some basic facts about RF, in Section 2. Then, Section 3 reviews some proposals about RF in parallel environments and highlights some questions. Section 4 develops a similar outline about online RF. Sections 5 to 7 contains remarks and proposals in these new contexts. Section 5 is devoted to the out-of-bag error and variable importance measure, Section 6 to sampling issues and Section 7 examines reweighting strategies. Then, Sections 8 to 10 are devoted to numerical experiments on two massive datasets, an extensive study on a simulated one and an application to a real world one. Finally, Section 11 concludes the paper.

2 Random Forests

Denoting by $L = \{(x_1, y_1), \dots, (x_n, y_n)\}$ a learning set of independent observations of the random vector (X, Y) , we distinguish $X = (X^1, \dots, X^p)$ where $X \in R^p$ is the vector of the predictors (or explanatory variables) from $Y \in \mathcal{Y}$ the explained variable, where Y is either a class label for classification problems or a numerical response for regression ones.

A classifier s is a mapping $s : R^p \rightarrow \mathcal{Y}$ while the regression function appears naturally to be the function s when we suppose that $Y = s(X) + \varepsilon$ with $E[\varepsilon|X] = 0$. RF provide estimators of either the Bayes classifier, which minimizes the classification error $P(Y \neq s(X))$ or the regression function.

RF are a learning method for classification and regression based on the CART (Classification and Regression Trees) method defined by Breiman *et al.* [22]. The left part of Figure 1 provides an example of classification tree. Such a tree allows to predict the class label corresponding to a given x -value by simply starting from the root of the tree (at the top of the figure) and by answering the questions until a leaf is reached. The predicted class is then the value labeling the leaf. Such a tree is equivalent to the classifier s . This classifier is the function which is piecewise constant on the partition described in the right part of Figure 1. Note that splits are parallel to the axes defined by the original variables leading to an additive model.

While CART is a well-known way to design optimal single trees by performing first a growing step and then a pruning one, the principle of RF is to aggregate many binary decision trees coming from two random perturbation mechanisms: the use of bootstrap samples of L instead of the whole sample L and the construction of a randomized tree predictor instead of CART on each bootstrap sample. The construction is summarized in Figure 2.

There are two main differences with respect to CART trees: first, in the growing step, at each node, a fixed number of input variables are randomly chosen and the best split is calculated only among them, and secondly, no pruning is performed.

In the next section, we will explain that most proposals made to adapt RF to Big Data often consider the original RF proposed by Breiman as an object that simply has to be mimicked in the Big Data context. But we will see, later on this article, that alternatives to this vision are possible. Some of these alternatives rely on other ways to resample the data and others are based on variants in the construction of the trees.

We will concentrate on the prediction performance of RF, focusing on out-of-bag (OOB) error, which allows to quantify the variable importance (VI in the sequel). The

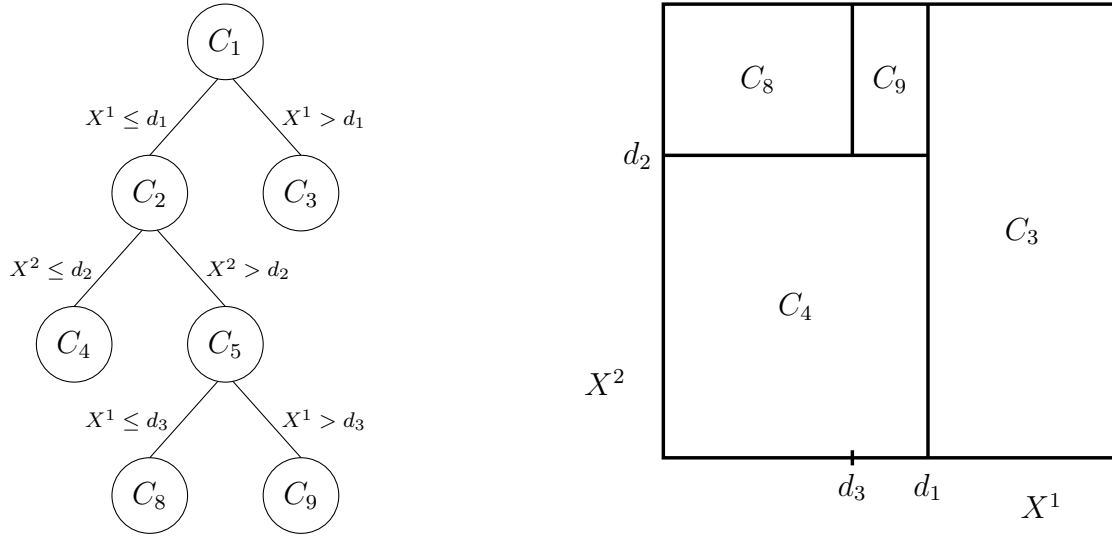


Figure 1: Left: a classification tree allowing to predict the class label corresponding to a given x -value. Right: the associated partition of the predictor space.

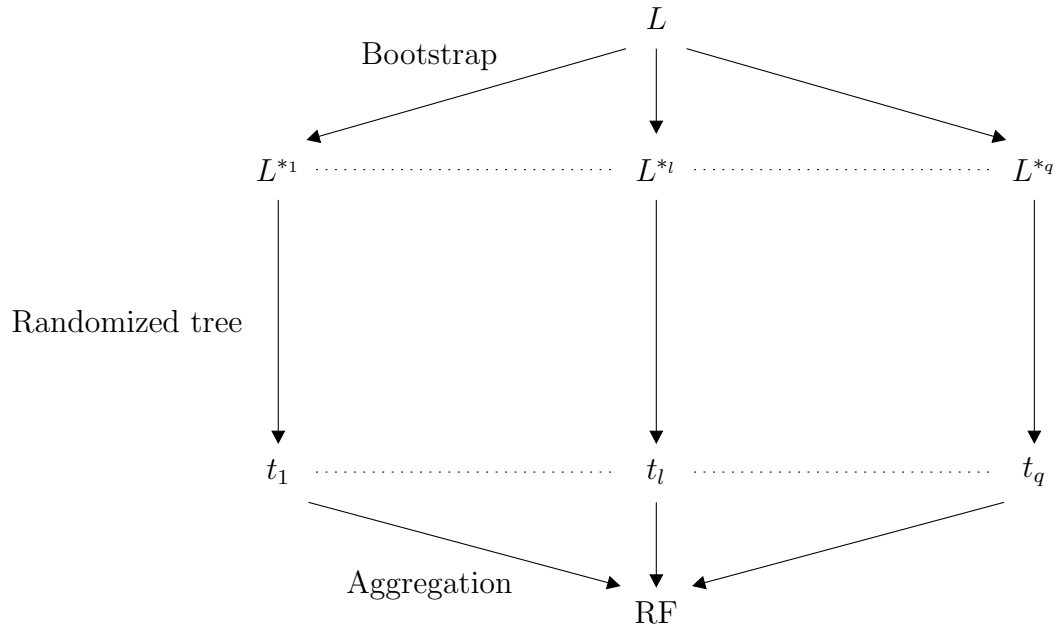


Figure 2: RF construction scheme: starting from L , generate bootstrap samples denoted by L^{*l} and construct corresponding randomized binary decision trees and finally aggregate them.

quantification of the variable importance is crucial for many procedures involving RF, *e.g.*, for ranking the variables before a stepwise variable selection strategy (see [23]).

For each tree t of the forest, consider the associated OOB_t sample (composed of data not included in the bootstrap sample used to construct t). The OOB error rate of the forest is defined, in the classification case, by:

$$\text{errForest} = \frac{1}{n} \text{Card} \{i \in \{1, \dots, n\} \mid y_i \neq \hat{y}_i\}$$

where \hat{y}_i is the most frequent label predicted by trees t for which i is in the associated OOB_t sample.

Denote by errTree_t the error (misclassification rate for classification) of tree t on its associated OOB_t sample. Now, randomly permute the values of X^j in OOB_t to get a perturbed sample denoted by $\widetilde{\text{OOB}}_t^j$ and compute $\text{err}\widetilde{\text{Tree}}_t^j$, the error of tree t on the perturbed sample. Variable importance of X^j is then equal to:

$$\text{VI}(X^j) = \frac{1}{\text{ntree}} \sum_t (\text{err}\widetilde{\text{Tree}}_t^j - \text{errTree}_t)$$

where the sum is over all trees t of the RF and ntree denotes the number of trees of the RF.

3 Random forests in scalable environments

3.1 Computing environments

Since RF are based on the definition of several independent trees, it is thus straightforward to obtain a parallel and faster implementation of the RF method, in which many trees are built in parallel on different cores.

However, computational time, which can be a bottleneck when dealing with large-size datasets and complex estimation procedures, is not the only challenge posed by the Big Data framework. First, as pointed out in [24], the notion of Big Data depends itself on the available computing resources: data can be considered as “large” if their size exceeds 20% of RAM and as “massive” if it exceeds 50% of RAM. This is especially true when relying on the free statistical software R [25], which capabilities are strictly limited by RAM.

In more extreme situations, the data are stored in distributed architectures, which means that the amount of available data is not only too large to be processed by a single processor (and does not fit in RAM). In this case, the data size can even be too large to fit in a single computer memory and the data are distributed among several computers. Frequently, the distribution is managed using specific frameworks dedicated to shared computing environments such as Hadoop⁴. For instance, Thusoo *et al.* [26] indicate that Facebook[©] had more than 21PB of data in 2010. Also, in this

⁴Hadoop, <http://hadoop.apache.org> is a software environment programmed in Java, which contains a file system for distributed architectures (HDFS: Hadoop Distributed File System) and dedicated programs for data analysis in parallel environments. It has been developed from GoogleFS, The Google File System.

situation, the data are frequently not structured data, properly indexed in a database and simple queries cannot be easily performed on such data. Hence, accessing a specific observation basically means reading all the data sequentially until this observation is reached.

3.2 MapReduce for RF

One approach to deal with such large-size data is to rely on a “divide-and-conquer” strategy. The large problem is divided into simpler subproblems and the solutions are gathered together to solve the original problem.

In particular, the MapReduce (MR) programming paradigm (see Figure 3) splits the data into small sub-samples, all processed independently in parallel. More precisely, MR proceeds in two steps: in a first step, called the Map step, the data set is split into several smaller chunks of data, $(x_i, y_i)_{i \in \tau_k}$, with $\cup_k \tau_k = \{1, \dots, n\}$ and $\tau_k \cap \tau_{k'} = \emptyset$, each one being processed by a separate core. These different Map jobs are independent and produce a list of couples of the form (key, value), where “key” is a key indexing the data that are contained in “value”. Then, in a second step, called the Reduce step, each reduce job proceeds all the outputs of the Map jobs that correspond to a given key value.

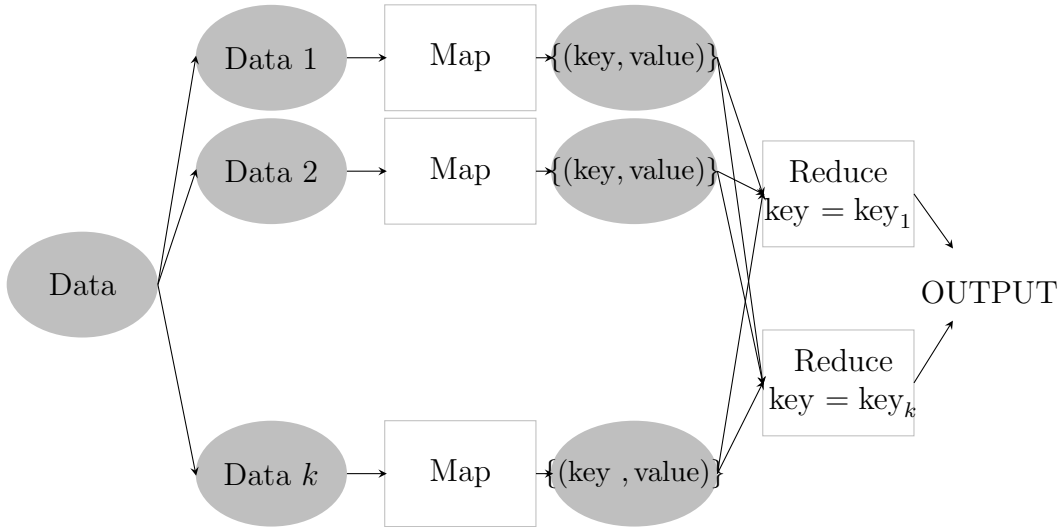


Figure 3: The MapReduce programming paradigm

In [12], Chu *et al.* show that many statistical methods can easily adapt to this paradigm: as long as the method is based on the computation of a sum $\sum_{i=1}^n [\dots]$, partial sums $S_d = \sum_{i \in \tau_d} [\dots]$ can be computed in the different Map jobs and the reduced step simply consists in computing the final sum $\sum_d S_d$. This approach can be directly applied to linear regression, naive Bayes, Gaussian discriminant analysis, ... and yields to estimates that are exactly the same as the one that would have been obtained with the direct global application of the method.

However, the situation is a bit different for RF, which are not based on sums. As indicated in [14], the standard MR version of RF, noted MR-RF in the sequel, is

based on the parallel construction of trees obtained on bootstrap *subsamples* of the data: each chunk of data is sent to an independent Map job in which a RF (which can have a moderate number of trees) is built (the output key is always equal to 1 and the output value is the forest). There is no reduce job as the output of the Map jobs is a collection of RF which, all merged together, give the final forest. The prediction itself, for all data, requires another MR pass, as explained in [14]. This method is the one implemented in the ApacheTM library Mahout.

3.3 Mismatches with original RF

This adaptation of RF to the MR framework is not strictly equivalent to the one that would have been obtained using a direct global approach. Indeed, data are thrown in the different Map jobs with no control on the representativity of the subsamples processed by each Map job. As noted by Laptev *et al.* [10], data are rarely ordered randomly in the Big Data world but rather, items clustered on some particular attributes are often placed next to each other on disk due to spatial locality. As a consequence, the samples sent to every different Map job might well be specific enough to produce very heterogeneous forests: there would be no meaning in simply averaging all those trees together to make a global prediction.

Another consequence of splitting the data before precessing every chunk independently is that, given that the Map jobs only have access to a (small) part of the data, the OOB error - and thus, the VI - cannot be obtained directly while the forest is trained. This issue is hard enough to overcome since it is generally too costly to record which data have been used to train which tree to re-compute the standard OOB error after the forest has been trained. In the standard MR-RF, an approximation of the OOB error is computed, as mentioned in Section 5, which is not equal to the OOB error of the MR-RF.

As pointed out by Kleiner *et al.* [9], another limit of the approach comes from the fact that each forest in a Map job is built on a subsample of the whole dataset. The success of m -out-of- n bootstrap samples (*i.e.*, sub-sampling with replacement to obtain a sample of size $m < n$, which is approximately what is done in the standard implementation of MR-RF algorithm) is highly conditioned on the choice of m which can not be well controlled or tuned within an MR implementation.

In this framework, to remain close to the performance of the standard bootstrap procedures, it is thus advised to keep m as close to n as possible. A consequence is thus that, even though m might be a size much more manageable for processing, it can still be very large. In the original algorithm from [18], the trees that composed the forest are fully developed trees, which means that the trees are grown until every terminal node (leaf) is perfectly homogeneous regarding the values of Y for the observations that falls in this node. When m is large, and especially in the regression case, this leads to very deep trees which all are computationally very expensive and even difficult to use for prediction purpose. However, as far as we know, no study addresses the question of the impact of controlling and/or tuning the maximum number of nodes in the forest's trees.

4 Online random forests

The general idea of online RF (ORF in the sequel), introduced by Saffari *et al.* [16], is to adapt RF methodology, in order to handle the case where data arrive sequentially (also known as a “data stream”). An online framework supposes that at time t one does not have access to all the data from the past, but only to the current observation. ORF are first defined in [16] and detailed only for classification problems. They combine the idea of online bagging from [27], Extremely Randomized Trees (ERT) from [28], and a mechanism to update the forest each time a new observation arrives.

More precisely, when a new data arrives, the online bagging updates k times a given tree, where k is sampled from a Poisson distribution to mimic a batch bootstrap sampling. This means that this new data will appear k times in the tree, which mimics the fact that one data can be drawn k times in the batch sampling (with replacement). ERT is used instead of original Breiman’s RF, because it allows for a faster update of the forest: in ERT, S splits (*i.e.*, a split variable *and* a split value) are randomly drawn for every node, and the final split is optimized only among those S candidate splits. Moreover, all decisions given by a tree are only based on the proportions of each class label among observations in a node. ORF keep up-to-date (in an online manner) an heterogeneity measure based on these proportions, used to determine the class label of a node. So when a node is created, S candidate splits (hence $2S$ candidate new nodes) are randomly drawn and when a new data arrives in an existing node, this measure is updated for all those $2S$ candidate nodes. This mechanism is repeated until a stopping condition is realized and the final split minimizes the heterogeneity measure among the S candidate splits. Then a new node is created and so on.

OOB error rate of a tree t is also estimated online: the current data is OOB for all trees for which the Poisson random variable used to replicate the observation in the tree, is equal to 0. The prediction provided for such a tree t is used to update errTree_t . However, as the prediction cannot be re-evaluated after the tree has been updated with next data, this approach is only an approximation of the original errTree_t .

From the theoretical viewpoint, the recent article [17] introduces a new variant of ORF. The two main differences with the original ORF are that, 1) no online bootstrap is performed. 2) The data stream is randomly partitioned in two streams: the structure stream and the estimation stream. Data from structure stream only participate on the splits optimization, while data from estimation stream are only used to allocate a class label to a node. Thanks to this partition, the authors manage to obtain consistency results of ORF.

Let us conclude this section, by giving several remarks on ORF. First, in the same manner as online update of errTree_t , it seems that an approximation of the OOB error of the entire forest, errForest , could be obtained online. However, permuting the values of a given variable when the data arrive in stream and are not stored after they have been processed is still an open issue for which [16, 17] give no solution. Hence, VI cannot be simply defined in the online framework. Second, the use of ERT instead of Breiman’s RF allows to reduce the computational cost. It would be of interest to use in MR-RF this RF variant, or even more randomized ones (like [29] PERT, Perfect Random Tree Ensembles, or [30, 31] PRF, Purely Random Forests). The idea of those latter variants is not to choose the variable involved in a split and

the associated threshold from the data but to randomly choose them according to different schemes. Finally, ORF could be use in the Big Data framework, using an update scheme, instead of building forest in parallel until all data have been processed, could be a way to use only a portion of the data set until the forest is accurate enough. Moreover, one valuable characteristic of ORF is that it could address both the issue of Volume and Velocity.

Despite their interest, ORF are no considered in the rest of the paper, due to the lack of available computer codes to carry out experimentations in a unified way involving R software.

5 Out-of-bag error and variable importance measure

In this section we briefly discuss some possible alternatives to overcome the limits of the existing solutions that we have described previously.

As explained in Section 2, OOB error and VI are important diagnostic tools to help the user understand the forest accuracy and to perform variable selection. However, these quantities may be unavailable directly (or in a standard manner) in the RF variants described in the two previous sections.

In Section 3, we explained that, due to the independence of the different Map jobs, the OOB error cannot be computed in the MR-RF variant: each forest trained in a Map job is unaware of the data which have not be sent to its map and the indices of the data that have been used to train a given forest are lost in the output of the Map jobs so there is no way to obtain OOB errors in the sense that is given in Section 2 (nor the global OOB error, errForest , but not even the tree specific OOB error, errTree_t). Because trees can not retain indices of the data they have used and can not access the data which are not in their chunk.

However, an approximation of those errors can be obtained restricting its computation to a given chunk of data and averaging all of them: using the notations that were previously introduced, an MR version of the global OOB error would be

$$\text{MRerrForest} = \frac{1}{n} \sum_k n_k \text{errForest}^k$$

where n_k is the cardinal of τ_k and $\text{errForest}^k = \frac{1}{n_k} \text{Card} \{i \in \tau_k | y_i \neq \hat{y}_i^k\}$ in which the predicted value \hat{y}_i^k is obtained by a majority vote rule on the trees built in the Map job of chunk τ_k for which i is OOB.

In a similar manner, the OOB error of tree t , built from chunk τ_k with n_k data items in it, is equal to:

$$\text{errTree}_t^k = \frac{1}{\text{Card}(\text{OOB}_t^k)} \text{Card} \{i \in \text{OOB}_t^k | y_i \neq \hat{y}_i^t\}$$

where OOB_t^k is the set of data in τ_k that are not included in the bootstrap sample used to build the tree t and \hat{y}_i^t is the prediction made by t for x_i . Similarly, using a

permutation of the values of X^j in the sample τ_k , $\widetilde{\text{errTree}}_t^{k,j}$ can be defined and a MR version of the VI would be:

$$\text{MR_VI}(X^j) = \frac{1}{qK} \sum_{k,t} \left(\widetilde{\text{errTree}}_t^{k,j} - \text{errTree}_t^k \right)$$

where q is the number of trees in every forest obtained from a given Map job and K is the number of Map jobs.

It would be interesting to investigate the difference between the estimates obtained with the standard version and the ones proposed for MR above. In particular, one could expect to observe a sensitivity to K (number of chunks) and to the possible bias in splitting the data.

6 Sampling and stratification

The first limit of MR-RF that has been pointed in Section 3 is the possible sampling bias coming from the split of data in several chunks for the parallel Map jobs. This could be addressed with a more careful design of the data partition, using at least a random partition of the data or, probably more efficiently, a stratification performed thanks to the explained variable Y .

Another point of view to address this issue, is to carefully use subsampling: some authors point the fact that using all data is probably not required to obtain accurate estimations. The first approach would be to select a subset of the observations at random but Meng [11] notes that random subsampling methods do not scale well themselves. Thus they propose a sampling method that can be used in parallel and is well suited to Big Data problems. An alternative is described by Laptev *et al.* in [10] who define an empirical approach (EARL which stands for Early Accurate Result Library), implemented in an MR framework, that allows to control the size of the subsample until the precision of the result is accurate enough. This approach allows to obtain a random subsample of the data, whereas Grover and Carey [32] propose an approach to obtain a subsample that satisfies a given predicate (*i.e.*, a given query), which could be used to obtain subsample distributed approximately as the dependent variable Y . The impact of such subsampling strategies is studied in some recent papers: some have focused on methods in which they obtain theoretical control over the difference in accuracy between training a given model on the whole dataset and on a subset of the data. For a clustering problem, [7, 8] both propose methods to summarize a large set of data by selecting a small subset of representative observations with fast approaches. They control the accuracy of the final result inferred to the whole dataset and its consistency compared to the result that would have been obtained if the whole dataset have been directly used. Using one of these subsampling strategies prior to RF or MR-RF should help overcome the problem of the splitting bias.

Another interesting alternative to subsampling and stratification is the BLB (Bag of Little Bootstrap) method described in [9]. This method aims at building bootstrap samples of size n , each one containing only $m \ll n$ different data. The size of the bootstrap sample is the classical one (n), thus avoiding the problem of the bias involved

by m -out-of- n bootstrap methods which are a direct consequence of the MR strategy (each chunk contains a portion of the dataset and thus leads to bootstrap sample having size m). The processing of this sample is simplified by a smart weighting scheme and is thus manageable even for very large n because it contains only a small fraction (m/n) of unique observations from the original data set. Interestingly, this approach is well supported by theoretical results because the authors prove its equivalence with the standard bootstrap method.

7 Reweighting sub-forests

Following a notation from Breiman (2001), RF lead to better results when there is a higher diversity among the trees of the forest. So recently, some extensions of RF have been defined for improving an initial RF. In [33], Fawagreh *et al.* use an unsupervised learning technique (Local Outlier Factor, LOF) to identify diverse trees in the RF and then, they perform ensemble pruning by selecting trees with the highest LOF scores to produce an extension of RF termed LOFB-DRF, much smaller in size than RF and performing better. This scheme can be extended by using other diversity measures, see [34] presenting a theoretical analysis on six existing diversity measures.

Another possible variant would be to consider the whole forest as an ensemble of forests and to adapt the majority vote scheme with weights that address, *e.g.*, the issue of the sampling bias. Recently in [35], Winham *et al.* propose to introduce a weighted RF approach to improve predictive performance: the weighting scheme is based on the individual performance of the trees and could be adapted to the MR-RF framework.

Along the same ideas and at this exploratory stage, we prefer to adapt simple and robust ideas coming from the variants of AdaBoost [36] for classification boosting algorithms. Recall that the basic idea is to generate many different base predictors obtained by perturbing the training set and to combine them. Each predictor is designed with a sample obtained from the original one by adaptive resampling, highlighting sequentially the observations poorly predicted. If it is difficult to use this scheme, even in the context of online forests since it needs to update the adaptive bootstrap distribution, in the context of RF for Big Data. But the aggregation part of the algorithm provides directly a way to weight individual sub-forests. Indeed, the l -th predictor is constructed on L^{*l} under the resampling distribution but the final performance is evaluated on the whole learning sample L according to a weighted combination of individual predictors, weighting tree t by $\alpha_t = 1/2 \ln(\epsilon_t/(1 - \epsilon_t))$ where ϵ_t is the misclassification error computed on L . This can be adapted by considering weighted forests using weights of similar form, evaluated on a same single subset of observations.

8 Three RF methods for Big Data in action

The present section is devoted to numerical experiments on a massive simulated dataset (15 millions of observations), which aim at illustrating and comparing three variants involving subsampling, Big Data-bootstrap and MapReduce. The next section (Section 9) specifically focuses on the influence of biases in subsampling, using the same

dataset and the same three methods. Finally, we analyze the performance obtained on a well-known real-world benchmark for Big Data experiments that contains airline on-time performance data in Section 10.

All experiments have been conducted on the same server (with concurrent access), with 8 processors AMD Opteron 8384 2.7Ghz, with 4 cores each, a total RAM equal to 256 Go and running on Debian 8 Jessie. To allow fair comparisons between the methods and to make them independent from a particular software framework or a particular programming language, all methods have been programmed with R (version 3.2.0), using the following packages:

- the package **readr** [37] (version 0.1.1), which allows to read more efficiently flat and tabular text files from disk;
- the package **randomForest** [38] (version 4.6-10), which implements RF algorithm using Breiman and Cutler’s original Fortran code;
- the package **parallel** [25] (version 3.2.0), which is part of R and supports parallel computation.

The aims of the simulations were multiple: firstly, different approaches designed to handle Big Data with RF were compared. The comparison was made on the point of view of the computational effort needed to train the classifier and also in term of its accuracy. Secondly, the differences between the OOB error estimated by standard methods corresponding to a given approach (which generally uses only a part of the data to be computed) was compared to the OOB error of the classifier estimated on the whole data set. Finally, focusing on the Map-Reduce approach, the impact of the representativity of the data was assessed, both in terms of the values of variable to predict and in terms of the underlined model.

To address all these issues, simulated data are studied in this section. They correspond to a well controlled model and can thus be used to obtain comprehensive results on the various questions described above. The simulated dataset corresponds to 15,000,000 observations generated from the model described in [39]: this model is an equiprobable two class problem in which the variable to predict, Y , takes values in $\{-1, 1\}$ and the predictors are, for 6 of them, true predictors, whereas the other ones (in our case only one) are noise. The simulation model is defined through the law of Y ($P(Y = 1) = P(Y = -1) = 0.5$) and the conditional distribution of the $(X^j)_{j=1,\dots,7}$ given $Y = y$:

- with probability equal to 0.7, $X^j \sim \mathcal{N}(jy, 1)$ for $j \in \{1, 2, 3\}$ and $X^j \sim \mathcal{N}(0, 1)$ for $j \in \{4, 5, 6\}$;
- with probability equal to 0.3, $X^j \sim \mathcal{N}(0, 1)$ for $j \in \{1, 2, 3\}$ and $X^j \sim \mathcal{N}((j - 3)y, 1)$ for $j \in \{4, 5, 6\}$;
- $X^7 \sim \mathcal{N}(0, 1)$.

All variables are centered and scaled to unit variance after the simulation process, which gave a dataset which size (in plain text format) was equal to 1.9 Go. Compared to the size of available RAM, this dataset was relatively moderate which allowed us to

perform extensive comparisons while being in the realistic Big Data framework with a large number of observations. With the **readr** package, loading this dataset took approximately one minute. As a baseline for comparison, a standard RF with 100 trees was trained in a sequential way with the R package **randomForest**. Considering the very large number of observations, the number of terminal nodes was limited to 500. The training of this forest took approximately 7 hours and the resulting OOB error was equal to $4.564e^{-3}$.

We designed experiments to compare this sequential forest to other approaches frequently used in the Big Data framework: *i)* using a sub-sample of the original data set to train a forest; *ii)* Bag of Little Bootstrap (BLB-RF), as described in [9]; *iii)* a Map-Reduce (MR-RF) decomposition of the training, as described in Section 3.2. The methods were all programmed in parallel with 10 cores. A varying number of parallel jobs and a varying number of trees in the forests were tested. Also, when applicable (*i.e.*, for the approach based on a sub-sample and for BLB), the number of distinct observations from the original dataset that were used to train the forest was varying. Similarly, for the MR approach, the number of chunks launched in parallel decides on the size of the sample sent to the chunk (*i.e.*, it is equal to the number of observations in the dataset divided by the number of chunks), which was thus also tested with different values. For this section, the purpose was only to compare the methods themselves so all subsamplings were done in such a way that the subsamples (or the subsamples sent to the different Map jobs) were representative of the whole dataset from the distributional viewpoint. Note that such a representative subsample is not always straightforward to obtain in real Big Data situations.

The different results are compared through the computational time needed by every method (real elapsed time as returned by R) and the prediction performance. This last quantity was assessed in different ways:

- i)* for the method using a sub-sample, the OOB error was calculated on the sub-sample and the OOB error was also calculated using the whole dataset (hence all observations outside the sub-sample were considered out-of-bag);
- ii)* for BLB-RF, the same approach was used: the OOB error was calculated on the subsample and on the whole original dataset;
- iii)* for MR-RF, the OOB error was calculated as described in Section 5 (*i.e.*, by averaging the OOB errors obtained in every Map job: MRerrForest). Also, keeping record of which observations have been sent to every map, we were able to compute the OOB error on the whole dataset (all observations not affected to a Map were considered out-of-bag for every trees trained in this map: errForest, as described in Section 2).

Additionally, a test sample, with 150,000 observations, was generated independently of the training sample and used to provide a reliable test error (misclassification rate) for all methods. More precisely, three types of errors were computed: BDerrForest, which is the OOB error as would be naturally computed for the corresponding method. This error is MRerrForest for MR-RF and it uses a subsample of the dataset in the other cases. The second error will be denoted by errForest and corresponds to the OOB error calculated by using the whole dataset. Finally, errTest denotes the test error.

Method	Computational time	BDerrForest	errForest	errTest
sampling 10%	3 min	4.622e(-3)	4.381e(-3)	4.300e(-3)
sampling 1%	9 sec	4.586e(-3)	4.363e(-3)	4.400e(-3)
sampling 0.1%	1 sec	5.600e(-3)	4.714e(-3)	4.573e(-3)
sampling 0.01%	0.3 sec	4.666e(-3)	5.957e(-3)	5.753e(-3)
BLB-RF 5/20	1 min	4.138e(-3)	4.294e(-3)	4.267e(-3)
BLB-RF 10/10	3 min	4.138e(-3)	4.278e(-3)	4.267e(-3)
MR-RF 100/1	2 min	1.397e(-2)	4.235 e(-3)	4.006e(-3)
MR-RF 100/10	2 min	8.646e(-3)	4.155e(-3)	4.293e(-3)
MR-RF 10/10	6 min	8.501e(-3)	4.290e(-3)	4.253e(-3)
MR-RF 10/100	21 min	4.556e(-3)	4.249e(-3)	4.260e(-3)

Table 1: Performance (computational time and misclassification rates) obtained by three different RF methods for Big Data. Method names and parameters are given in the first column (sampling fraction, K/q for the other two: K parallel processes and q trees). OOB error as estimated by the model is given in the third column and OOB error calculated using the whole dataset is given in the fourth column. The last column contains a misclassification rate calculated on an independent test sample.

Results are given in Table 1. For these simulations, the subsampling approach was trained with a varying sampling fraction (10%, 1%, 0.1% or 0.01%) of observations taken at random from the original dataset. The forests all contained 100 trees and were trained in parallel using 10 processes (each training 10 trees) on 10 cores. The BLB-RF was performed using K different samples ($K = 5$ or 10) of about 15,000,000^{0.7} distinct observations (about 0.7% of the original dataset) each and for every sample, a forest with q trees ($q = 20$ or 10) was trained using a reweighting scheme as described in [9]. K cores were used to train the different forests in parallel. The parameters K and q are provided in the method name in the first column of Table 3. Finally, MR-RF was performed using K ($K = 100$, or 10) Map jobs with a forest of q trees ($q = 1$, 10 or 100) trained in every Map job. 10 cores were used to perform the different Map jobs in parallel. The parameters K and q are also provided in the method name in the first column of Table 3. In short, the results are quite comparable since they are based on forests containing 100 trees (except for some of the MR-RF results) and that were learned in parallel on 10 cores. In all simulations, the maximal number of leaves in the trees was set to 500.

Several conclusions can be driven from these results. First, the computational time needed to train all these Big Data versions of RF is almost the same and quite reduced (about a few minutes) compared to the sequential RF. The fastest approach is to extract a very small subsample and the slowest is the MR-RF approach with 10 maps of 100 forests each (because the number of observations sent to each Map job is not much reduced compared to the original dataset). The results are not shown for the sake of simplicity but the performances are also quite stable: when a method was trained several times with the same parameters, the performances were almost always very close.

Regarding the errors, it has first to be noted that the generalization error (as assessed with errTest) is much better estimated by errForest than by the proxy of

the OOB error provided by the corresponding method, BDerrForest. In particular, BDerrForest tends to underestimate the error for the subsampling approaches when the subsample size is very small and to overestimate it (sometimes strongly) for MR-RF.

Finally, many methods achieve a performance which is quite close to that of the standard RF algorithm: the subsampling approach is quite close to the original algorithm, even for very small subsamples (with at least 0.1% of the original observations, the difference between the two predictors is not very important). The BLB approach is also very close to the original algorithm and remarkably stable to a change in its parameters K and q . Finally, MR-RF also gives an accurate predictor but its estimated OOB is close to the generalization error only when the number of Map jobs is small and the number of trees in the forest large enough: this is obtained at the price of a higher computational cost (about 10 times larger than for the other approaches).

9 More about subsampling biases and tree depth

In the previous section, simulations were conducted with representative subsamples and a maximal number of terminal leaves equal to 500 for every trees in every forest. The present section pushes the analysis a bit further by specifically investigating the influence of these two features on the results. All simulations were performed with the same dataset and the same computing environment than in the previous section.

As explained in Section 3.2, MR-RF can be influenced by the lack of representativeness of the data sent to the different Map jobs. In this section, we have used the same simulated dataset as above to address this issue and to evaluate the influence of such cases in two different directions:

- firstly, we have considered the *non representativity of the Y values in the different Map jobs* by artificially creating highly unbalanced chunks: this was done by
 - i) creating chunks with a varying proportion of observations coming from the first class (*i.e.*, such that $Y = 1$). More precisely, some of the chunks were highly unbalanced with respect to Y (*e.g.*, about 0.01%, 0.1%, 99.99% or 99.9% of the observations in those chunks came from the first class), some of the chunks were moderately unbalanced (*e.g.* about 10%, 25%, 75% or 90% of the observations coming from the first class) and some were balanced (about 50% of the observations coming from the first class). This first approach is denoted by “unbalanced1” in the sequel;
 - ii) creating unbalanced chunks that all contain the same proportion of one of the two classes, *i.e.*, 50% of the chunks contain a proportion p (with $p \in \{10; 1\}\%$) of observations coming from the first class ($Y = 1$). This approach is denoted by “unbalanced2” in the following;
- secondly, we addressed the *non representativity of the X values in the different Map jobs*: most chunks (about three fourth of the chunks) were designed to contain observations corresponding to only one of the two submodels, either $X^j \sim \mathcal{N}(jy, 1)$ for $j \in \{1, 2, 3\}$ and $X^j \sim \mathcal{N}(0, 1)$ for $j \in \{4, 5, 6\}$ or $X^j \sim \mathcal{N}(0, 1)$ for $j \in \{1, 2, 3\}$ and $X^j \sim \mathcal{N}((j-3)y, 1)$ for $j \in \{4, 5, 6\}$. This simulation

Method	Comp. time	BDerrForest	errForest	errTest
unbalanced1 100/1	3 minutes	3.900e(-3)	5.470e(-3)	5.247e(-3)
unbalanced1 10/10	8 minutes	2.575e(-3)	4.714e(-3)	4.473e(-3)
unbalanced2 100/1/0.1	2 minutes	5.880e(-3)	4.502e(-3)	4.373e(-3)
unbalanced2 10/10/0.1	3 minutes	4.165e(-3)	4.465e(-3)	4.260e(-3)
unbalanced2 100/1/0.01	1 minute	1.926e(-3)	8.734e(-2)	4.484e(-2)
unbalanced2 10/10/0.01	4 minutes	9.087e(-4)	7.612e(-2)	7.299e(-2)
x-biases 100/1	3 minutes	3.504e(-3)	1.010e(-1)	1.006e(-1)
x-biases 100/10	3 minutes	2.082e(-3)	1.010e(-1)	1.008e(-1)

Table 2: Performance (computational time and misclassification rates) obtained by MR-RF in presence of subsampling biases in the different Map jobs. Method names and parameters are given in the first column (K/q for unbalanced1 and x-biases and $K/q/p$ for unbalanced2: K parallel processes, q trees and p fixed proportion of observations coming from one class in the chunks). OOB error as estimated by the model is given in the third column and OOB error calculated using the whole dataset is given in the fourth column. The last column contains a misclassification rate calculated on an independent test sample.

corresponds to the case where the Map jobs produce very heterogeneous forests. This issue has been discussed in Section 3.3 and we will show that it indeed has a strong impact in practice. These simulations will be denoted by “x-biases” in the sequel.

Finally, MR-RF was performed using K different chunks ($K = 10$ or 100) with a forest of q trees trained by every Map job ($q = 1, 10$ or 100). All Map jobs were trained using 10 cores in parallel. Similarly as in the previous case, the trees were all restricted to have at most 500 leafs. The results are provided in Table 2.

The first fact worth noting in these results is again that the OOB error as estimated by the MR-RF is not a good proxy for the generalization error of the forest: compared to the OOB error re-calculated with all data and to the test error, it often gives a very optimistic evaluation of the performance of the forest. This underestimation of the generalization error rate is even stronger than in the standard case because it is estimated in every Map job which is designed to train a good forest for the (biased) data that it has received: it is expected that the OOB error calculated in a given Map job is not so bad, whereas the forest obtained is probably a poor predictor for the entire dataset.

The second conclusion of these experiments is that Map jobs have to receive data that are strongly unbalanced in Y so as to deteriorate the performance of MR-RF (case unbalanced2 $p = 1\%$, which correspond to the case where all Map jobs have a given ratio $p = 1\%$ of observations coming from one of the two classes). In all the other cases, unbalanced Y values in Map job only slightly deteriorate the global performance of the forest. A similar conclusion is obtained for biases towards X values: the performance of the forest is strongly deteriorated and test misclassification rate is multiplied by a factor of more than 50.

As mentioned above, in Section 8, the maximal number of leaves was set to 500 in

Sampling fraction	Comp. time	Max. tree size	Pruned tree size	mean Gini
100%	5 hours	60683	3789	0.233
10%	13 min	6999	966	0.183
1%	23 sec	906	187	0.073
0.1%	0.01 sec	35	10	0.000

Table 3: Number of leaves and leaves heterogeneity of trees built on various fractions of data. Second column indicates computational time needed to built one tree, while number of leaves of the maximal tree and the optimal pruned tree are given in third and fourth column respectively. The last column the mean Gini index over all leaves of a tree and over 100 trees.

order to get comparable tree complexities. However homogeneity (in terms of classes) of leaves differs when a tree is built on the entire dataset or on a fraction of it. To illustrate this we computed the mean (over all leaves of a tree and over 100 trees) Gini index (defined by $2\hat{p}(1 - \hat{p})$, with \hat{p} the proportion of observations of class 1). Results are reported in Table 3.

For the 0.1% sampling fraction, tree leaves are pure (*i.e.*, contain observations all of the same class) and the 1% sampling fraction case is quite similar. But for the 100% and 10% cases, heterogeneity of leaves is more important. Thus, we investigated the effect of trees depth on RF performance. Recall that in RF all trees are typically grown to maximal trees (splits are performed until each leaf is pure) and that in CART an optimal tree is obtained by pruning the maximal tree. Table 3 contains the number of leaves of the maximal tree and the optimal CART tree associated to each fraction of data. Trees with 500 leaves are very far from maximal trees in most cases and even far from optimal CART tree for 100% and 10% sampling fraction. We finally computed performance of 5 RF methods using maximal trees instead of 500 leaves trees. The results are collected in Table 4.

Method	Computational time	errTest
standard	8 hours	3.980e(-3)
sampling 10%	4 min	3.933e(-3)
sampling 1%	10 sec	4.313e(-3)
MR-RF 100/1	2 min	4.033e(-2)
MR-RF 10/10	4 min	3.987e(-3)

Table 4: Performance (computational time and misclassification rates) obtained by different RF methods for Big Data using maximal trees. Names of the methods are given in the same manner as in Table 1 while "standard" refers to classical Breiman RF in a sequential way.

Computational times are actually comparable to those in Table 1, while misclassification rates are slightly better. The remaining heterogeneity when developing trees with 500 leaves does not affect much the performance in that case. Hence, while pruning all trees leads to prohibitive computational time, a constraint on tree size may be well adapted to the Big Data case. This point needs a more in-depth analysis and is left for further research.

10 Airline dataset

In the present section, similar experiments are performed with a real world dataset related to flight delays. The data were first processed in [24] to illustrate the use of the R packages for big data computing **bigmemory** and **foreach** [40]. In [24], the data were mainly used for description purpose (*e.g.*, quantile calculation), whereas we will be using it for prediction. More precisely, five variables based on the original variables included in the data set were used to predict if the flight was likely to arrive on time or with a delay larger than 15 minutes (flights with a delay smaller than 15 minutes were considered on time). The predictors were: the moment of the flight (two levels: night/daytime), the moment of the week (two levels: weekday/week-end), the departure time (in minutes, numeric) and distance (numeric). The dataset used to make the simulations contained 120,748,239 observations (observations with missing values were filtered out) and had a size equal to 3.2 GB (compared to the 12.3 GB of the original data with approximately the same number of observations). Loading the dataset and processing it to compute and extract the predictors and the target variables took approximately 30 minutes. Another feature of the dataset is that it is unbalanced: most of the flight are on time (only 19.3% of the flights are late).

The same method than the one described in Section 8 were compared:

- a standard RF was computed sequentially. It contained 100 trees. The RF took 16 hours to be obtained and its OOB error was equal to 18.32%;
- RF trained with a subsample of the total data (10%, 1%, 0.1% and 0.01% of all the observations were sampled at random without replacement). These RF were trained in parallel with 15 cores, each core building 7 trees from bootstrap samples coming from the common subsample (the final RF hence contained 105 trees);
- a RF was also trained using the BLB approach: it used $K = 15$ subsamples, each containing about 454,272 observations (about 0.4% of the size of the total data set. 15 subforests were trained in parallel with 7 trees each: the final forest hence contained 105 trees);
- MR-RF were also obtained with varying number of Map jobs ($K = 15$ or $K = 100$) and a varying number of trees in the forest grown in the different maps ($q = 7, 10$ or 20). The final RF contained from 105 to 1000 trees.

In all methods, the maximum number of terminal leafs in the trees was set to 500 and all RF were trained in parallel on 15 cores, except for the sequential approach. Results are given in Table 5 in which the notations are the same as in Section 8. The results show that there is almost no difference in terms of performance accuracy between using all data and using only a small proportion (about 0.01%) of them. In terms of compromise between computational time and accuracy, using a small subsample is clearly the best strategy, provided that the user is able to obtain a representative subsample at a low computational cost. Also, contrary to what happened in the example described in Section 9, BDerrForest is always a good approximation of errForest.

Method	Computational time	BDerrForest	errForest
sampling 10%	32 min	18.32%	18.32%
sampling 1%	2 min	18.35%	18.33%
sampling 0.1%	7 sec	18.36%	18.39%
sampling 0.01%	2 sec	18.44%	18.49%
BLB-RF 15/7	25 min	18.35%	18.33%
MR-RF 15/7	15 min	18.33%	18.27%
MR-RF 15/20	25 min	18.34%	18.20%
MR-RF 100/10	17 min	18.33%	18.20%

Table 5: Performance (computational time and misclassification rates) obtained by three different RF methods for Big Data. Method names and parameters are given in the first column (sampling fraction, K/q for the other two: K parallel processes, q trees). OOB error as estimated by the model is given in the third column and OOB error calculated using the whole dataset is given in the fourth column.

In addition, the impact of the representativity, with respect to the target variable, of the samples on which the RF were trained was assessed: instead of using a representative (hence unbalanced) sample from the total dataset, a balanced subsample (for 50% of delayed flights and 50% of on time flights) was obtained and used as the input data to train the random forest. Its size was equal to 10% of the total dataset size. This approach obtained an errForest equal to 33.34% (and BDerrForest was equal to 39.15%), which is strongly deteriorated compared to the previous misclassification rates. In this example, the representativity of the observations contained in the subsample strongly impacts the estimated model. The model with balanced data has a better ability to detect late flights and favors the sensitivity over the specificity.

11 Conclusion

This paper aims at extending standard Random Forests in order to process Big Data that exceed the memory and computing resources of a single computer. Indeed RF is an interesting example among the widely used statistical methods in machine learning since it already offers several ways to deal with massive data and data streams.

Focusing on classification problems, we reviewed some of the available proposals about RF in parallel environments and online RF. We formulated various remarks for RF in the Big Data context, including out-of-bag type errors and related variable importance measures.

We experimented on two massive datasets (15 and 120 millions of observations), a simulated one and real world data, three variants involving subsampling, BLB and MapReduce respectively. The simulations showed that the Big Data versions of RF all yield to a reduced computational time with no loss in accuracy. In particular, the subsampling and BLB approaches, which use only a very small amount of the original dataset are very fast and give similar performances to a sequential RF trained on the whole dataset. The only drawback of these approaches is in the estimation of the OOB error which seems to be sometimes badly estimated by the OOB error calculated on

the subsample. However, when the amount of data is that big, it is easy to calculate an error on an independent small test subsample, at low computational cost.

Finally, the most crucial point stressed in the simulations is that the lack of representativeness of subsamples can result in drastic deterioration of MR-RF performance. However, designing a subsample representative enough of the whole dataset is an open issue *per se* in the Big Data context but out of the scope of the present article. Alternatively, we suggest that a relevant reweighting scheme could be used to address this issue for BD-RF.

References

- [1] J. Fan, F. Han, and H. Liu, “Challenges of big data analysis,” *National Science Review*, vol. 1, no. 2, pp. 293–314, 2014.
- [2] R. Hoerl, R. Snee, and R. De Veaux, “Applying statistical thinking to ‘Big Data’ problems,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 6, no. 4, pp. 222–232, 2014.
- [3] M. Jordan, “On statistics, computation and scalability,” *Bernoulli*, vol. 19, no. 4, pp. 1378–1390, 2013.
- [4] P. Besse, A. Garivier, and J. Loubes, “Big data - Retour vers le futur 3. De statisticien à data scientist.” arXiv preprint arXiv:1403.3758, 2014.
- [5] P. Besse and N. Villa-Vialaneix, “Statistique et big data analytics. Volumétrie, l’attaque des clones.” arXiv preprint arXiv:1405.6676, 2014.
- [6] C. Wang, M. Chen, E. Schifano, J. Wu, and J. Yan, “A survey of statistical methods and computing for big data.” arXiv preprint arXiv:1502.07989, 2015.
- [7] M. Bădoiu, S. Har-Peled, and P. Indyk, “Approximate clustering via core-sets,” in *Proceedings of the 34th annual ACM Symposium on Theory of Computing* (J. Reif, ed.), no. 250-257, (Montreal, QC, Canada), ACM New York, NY, USA, 2002.
- [8] D. Yan, L. Huang, and M. Jordan, “Fast approximate spectral clustering,” in *Proceedings of the 15th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining* (J. Elder, F. Soulié-Fogelman, P. Flach, and M. Zaki, eds.), pp. 907–916, ACM New York, NY, USA, 2009.
- [9] A. Kleiner, A. Talwalkar, P. Sarkar, and M. Jordan, “A scalable bootstrap for massive data,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 76, no. 4, pp. 795–816, 2014.
- [10] N. Laptev, K. Zeng, and C. Zaniolo, “Early accurate results for advanced analytics on mapreduce,” in *Proceedings of the 28th International Conference on Very Large Data Bases*, vol. 5 of *Proceedings of the VLDB Endowment*, (Istanbul, Turkey), 2012.
- [11] X. Meng, “Scalable simple random sampling and stratified sampling,” in *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, vol. 28 of *JMLR: W&CP*, (Georgia, USA), 2013.
- [12] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun, “Map-Reduce for machine learning on multicore,” in *Advances in Neural Information Processing Systems (NIPS 2010)* (J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), vol. 23, (Hyatt Regency, Vancouver, Canada), pp. 281–288, 2010.
- [13] X. Chen and M. Xie, “A split-and-conquer approach for analysis of extraordinarily large data,” *Statistica Sinica*, vol. 24, pp. 1655–1684, 2014.
- [14] S. del Rio, V. López, J. Benítez, and F. Herrera, “On the use of MapReduce for imbalanced big data using random forest,” *Information Sciences*, vol. 285, pp. 112–137, 2014.

- [15] V. Chandrasekaran and M. Jordan, “Computational and statistical tradeoffs via convex relaxation,” *Proceedings of the National Academy of Sciences USA*, vol. 13, pp. E1181–E1190, 2013.
- [16] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, “On-line random forests,” in *Proceedings of IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 1393–1400, IEEE, 2009.
- [17] M. Denil, D. Matheson, and N. de Freitas, “Consistency of online random forests,” in *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, pp. 1256–1264, 2013.
- [18] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [19] E. Scornet, G. Biau, and J. Vert, “Consistency of random forests,” *The Annals of Statistics*, vol. 43, no. 4, pp. 1716–1741, 2015.
- [20] A. Verikas, A. Gelzinis, and M. Bacauskiene, “Mining data with random forests: a survey and results of new tests,” *Pattern Recognition*, vol. 44, no. 2, pp. 330–349, 2011.
- [21] A. Ziegler and I. König, “Mining data with random forests: current options for real-world applications,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 1, pp. 55–63, 2014.
- [22] L. Breiman, J. Friedman, R. Olsen, and C. Stone, *Classification and Regression Trees*. New York, USA: Chapman and Hall, 1984.
- [23] R. Genuer, J. Poggi, and C. Tuleau-Malot, “Variable selection using random forests,” *Pattern Recognition Letters*, vol. 31, no. 14, pp. 2225–2236, 2010.
- [24] M. Kane, J. Emerson, and S. Weston, “Scalable strategies for computing with massive data,” *Journal of Statistical Software*, vol. 55, 11 2013.
- [25] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.
- [26] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, “Data warehousing and analytics infrastructure at facebook,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2010)*, pp. 1013–1020, 2010.
- [27] N. Oza, “Online bagging and boosting,” in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2340–2345, IEEE, 2005.
- [28] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.
- [29] A. Cutler and G. Zhao, “Pert-perfect random tree ensembles,” *Computing Science and Statistics*, vol. 33, pp. 490–497, 2001.
- [30] G. Biau, L. Devroye, and G. Lugosi, “Consistency of random forests and other averaging classifiers,” *The Journal of Machine Learning Research*, vol. 9, pp. 2015–2033, 2008.
- [31] S. Arlot and R. Genuer, “Analysis of purely random forests bias.” arXiv preprint arXiv:1407.3939, 2014.
- [32] R. Grover and M. Carey, “Extending map-reduce for efficient predicate-based sampling,” in *Proceedings of IEEE International Conference on Data Engineering (ICDE 2012)*, (Washington, DC, USA), pp. 486–497, 2012.
- [33] K. Fawagreh, M. Gaber, and E. Elyan, “An outlier detection-based tree selection approach to extreme pruning of random forests.” arXiv preprint arXiv:1503.05187, 2015.
- [34] E. Tang, P. Suganthan, and X. Yao, “An analysis of diversity measures,” *Machine Learning*, vol. 65, pp. 247–271, 2006.
- [35] S. J. Winham, R. Freimuth, and J. Biernacka, “A weighted random forests approach to improve predictive performance,” *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 6, no. 6, pp. 496–505, 2013.

- [36] Y. Freund and R. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.
- [37] H. Wickham and R. François, **readr**: *Read Tabular Data*, 2015. R package version 0.2.2.
- [38] A. Liaw and M. Wiener, “Classification and regression by randomForest,” *R News*, vol. 2, no. 3, pp. 18–22, 2002.
- [39] J. Weston, A. Elisseeff, B. Schoelkopf, and M. Tipping, “Use of the zero norm with linear model and kernel methods,” *Journal of Machine Learning Research*, vol. 3, pp. 1439–1461, 2003.
- [40] Revolution Analytics and S. Weston, **foreach**: *Foreach looping construct for R*, 2014. R package version 1.4.2.